DOMAIN SPECIFIC SOFTWARE ARCHITECTURES -- COMMAND AND CONTROL

Christine Braun
William Hatch
Theodore Ruegsegger
GTE Federal Systems
15000 Conference Center Dr.
Chantilly, VA 22021

Bob Balzer
Martin Feather
Neil Goldman
Dave Wile
USC/Information Sciences Institute
Marina Del Rey, CA 90292

Abstract

GTE is the Command and Control contractor for the Domain Specific Software Architectures program. The objective of this program is to develop and demonstrate an architecture-driven, component-based capability for the automated generation of command and control (C2) applications. Such a capability will significantly reduce the cost of C2 application development and will lead to improved system quality and reliability through the use of proven architectures and components.

A major focus of GTE's approach is the automated generation of application components in particular subdomains. Our initial work in this area has concentrated in the message handling subdomain; we have defined and prototyped an approach that can automate one of the most software-intensive parts of C2 systems development.

This paper provides an overview of the GTE team's DSSA approach and then presents our work on automated support for message processing.

The DSSA Concept

DSSA is based on the concept of an accepted generic software architecture for the target domain. As defined by DSSA, a software architecture describes the topology of software components, specifies the component interfaces, and identifies computational models associated with those components. The architecture must apply to a wide range of systems in the chosen domain; thus it must be general and flexible. It must be established with the consensus of practitioners in the domain.

Once an architecture is established, components that conform to the architecture—i.e., that implement elements of its functionality in conformance with its interfaces—will be acquired. They may be acquired by identifying and modifying (if required) existing components or by specifically creating them. One of the ways they may be created is through automated component generation. DARPA has sponsored work in this area at USC Information Sciences Institute -- the AP5 application generator project, and is interested in incorporating this or related technology.

The existence of a domain-specific architecture and conformant component base will dictate a significantly different approach to software application development. The developer will not wait until detailed design or implementation to search for reuse opportunities; instead, he/she will be driven by the architecture throughout. The architecture and component base will help define requirements and allow construction of rapid prototypes. Design will use the architecture as a starting point. Design and development tools will be automated to "walk through" the architecture and assist the developer in the selection of appropriate components. The ultimate goal is to significantly automate the generation of applications. A major DSSA task is to define such a software lifecycle model and to prototype a supporting toolset.

These activities will be accompanied by extensive interaction with the development community for the target domain, and by technology transition activities. One aspect of this is that each domain team is working closely with a DoD agency that carries out major developments in the designated area. The GTE team is working with the US Army Communications and Electronics Command.

Why Command and Control?

There are many reasons why the command and control domain is an excellent target for DSSA technology. It is

a high payoff area; command and control systems are needed even in the current military climate. (This is particularly true when one recognizes that applications such as drug interdiction fall within the C2 "umbrella".) It is a well-understood area; most of the processing performed in C2 applications is not algorithmically complex. However, C2 applications are very large, and much of this size comes from repeated similar processing -- for example, parsing hundreds of types of messages. In addition to this commonality within applications, there is much commonality across applications. Multiple C2 systems must handle the same message types, display the same kinds of world maps, etc.

The kinds of commonality in C2 applications are very well-suited to DSSA techniques. In some areas, components can be reused identically; these can be placed in the DSSA component base and highly optimized. In other areas, components will be very similar in nature but differ in the particulars, e.g., message parsing. These areas are a natural fit to the DSSA component generation technology, allowing a table-driven generator to quickly create the needed specific component instances.

GTE's Approach

Figure 1 illustrates GTE's overall approach to the DSSA program.

Initially, project work will follow two parallel threads. The first will define a software process model appropriate to architecture-driven software development and will develop a toolset to support that process. The second will establish a capability that implements the process for the command and control domain, based on a C2 architecture and a set of reusable C2 components.

The DSSA process model will address all aspects of the software life cycle. It will describe activities for establishing system requirements, developing the software system, and sustaining the system after delivery. The DSSA toolset will support all of these

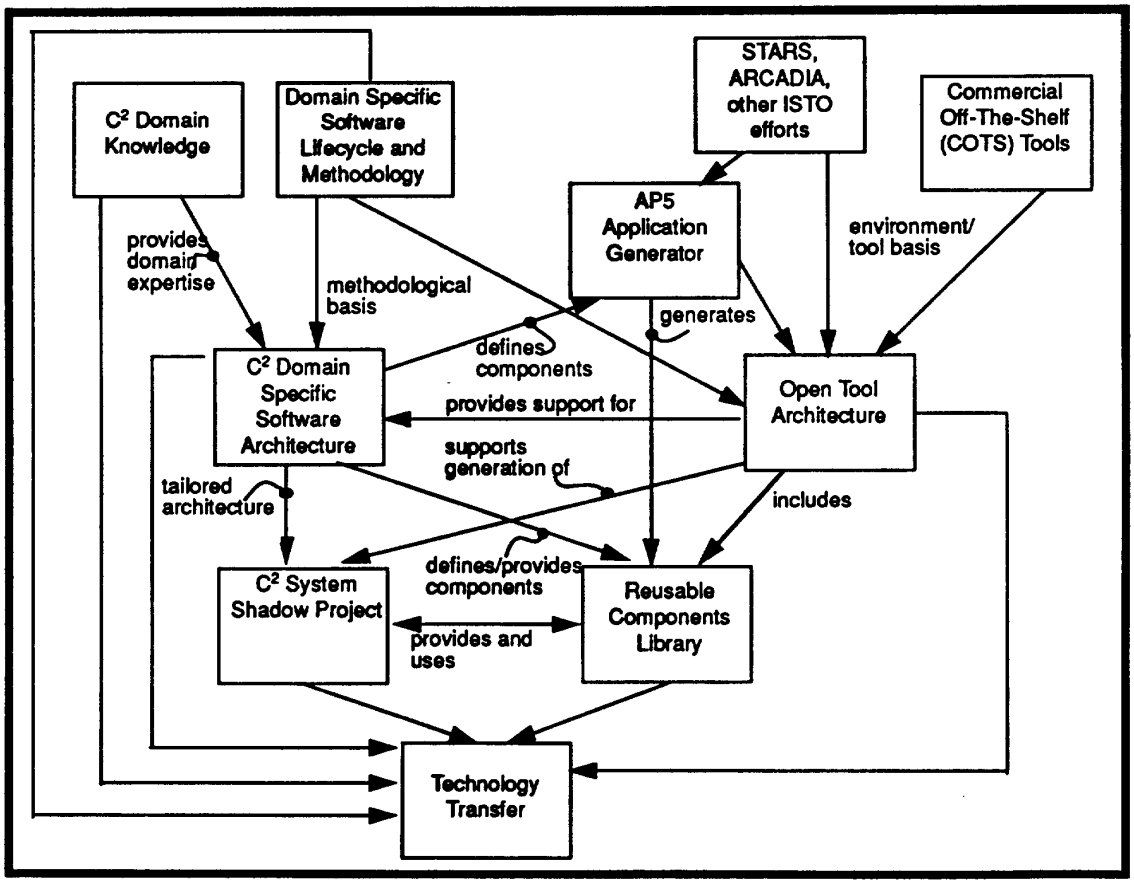

Figure 1. GTE's DSSA Approach

activities, automating them as far as possible. In particular, it will automate system development activities by using the architecture as a template, guiding the selection of available reusable components, and automating the generation of specific required components. The toolset will be constructed insofar as possible from available tools -- both commercial products and products of the research community. In particular, it will make use of USC/ISI's AP5 application generator, DARPA/ STARS reuse libraries, and DARPA/ Prototech tools. Open tool interfaces will be emphasized to minimize specific tool dependencies, thus making the toolset usable in the widest range of environments.

Fundamental to the C2 DSSA capability is the development of a C2 software architecture. This starts with development of a multi-viewpoint domain model, created through interaction with all elements of the DoD C2 community. The automated Requirements Driven Development (RDD) methodology will be used in model creation. From this, an object-oriented software architecture will be developed. The architecture will tie back to the multi-viewpoint model so that mappings to different views of the domain functional decomposition are apparent. George Mason University's Center for C3I will play a major part in this modeling and consensus-building activity. A base of components conforming to the architecture will then be developed. Many of these will be existing components, perhaps modified to fit the architecture. Others will be automatically generated using AP5.

The DSSA capability will be demonstrated by development of a prototype C2 system, most likely an element of the Army Tactical Command and Control System (ATCCS). An independent metrics/validation task will assess the effectiveness of the approach and gather metrics. The methodology and toolset will be revised based on findings and further necessary research will be identified.

Throughout the program, a technology transfer task will present results in conferences, papers, seminars, and short courses. The George Mason University Center for C3I will serve as a focal point for technology transfer.

Application Generation

The Technology

Application generators are tools that permit software developers to create software application programs in a much higher-level language tailored to the application domain. These programs are automatically translated by the application generator to a lower-level language, thus "generating applications." This greatly reduces the effort required to create working applications, typically by at least an order of magnitude. The benefits are analogous to those achieved by moving from assembly language development to use of standard procedural languages such as FORTRAN, C, and Ada.

Fourth Generation Languages (4GLs) are application generators for DBMS-oriented information system applications. Because 4GLs focus on a narrow class of applications, they can include very powerful constructs that allow software to be developed quickly and easily by those familiar with the application domain. Management Information System (MIS) developers using 4GLs achieve productivity improvements of as much as 50-100 times over traditional (usually COBOL) language users.

Application generators can be (and have been) developed for other types of applications as well. They are best suited to narrow domains, or subdomains of large domains such as C2. Because they require a domain specific vocabulary for expressing applications, they are generally unique to the domain or subdomain and not easily modified to handle other domains. Creation of an application generator for a particular domain, furthermore, is a significant undertaking. Development of an application generator is most appropriate in domains that are well-understood and in which many different developments perform primarily the same kinds of processing.

The AP5 Approach

USC Information Sciences Institute (ISI) has developed a capability (called AP5) that supports the development of application generators. AP5 is based on the concept of relational abstraction. The application developer identifies abstract data objects and the logical relationship among them. Effectively, the developer has access to a "virtual database" expressed succinctly in terms of the known structure of the domain's data model. Application behavior is then expressed in terms of these data objects, accessing them associatively via queries and modifying them based on values of other objects. This allows the user to concentrate on behavior rather than representation, and provides the power to express that behavior at a very high level.

Providing an AP5 application generator for a particular subdomain requires the development of a domain-specific language for that domain. This is a relatively straightforward task because the language, regardless of domain, involves the same fairly simple set of relation-oriented constructs for expressions data relationships, validations, and actions. It is also a critical task, because the expressive capability of this language is what provides the application generator's power. A translator

is then developed to map the language to an underlying program generator, which produces executable procedural code. This is also not too complex, as all languages contain similar constructs. Most of work is done by the underlying generator. (Currently the system generates LISP; an Ada generator is in development.)

A drawback to many existing application generators is poor efficiency of the generated code. This has, in many cases, made these generators suitable only for developing prototypes. AP5 addresses this problem by allowing the user to specify *annotations* that provide guidance to the translator on desired implementations of specific operations. These annotations can be added incrementally while tuning to achieve desired performance.

AP5 can play a key role in the C2 DSSA program. We anticipate that a number of C2 subdomains will be amenable to this approach. By developing generators for those subdomains we can achieve two major advances in productivity:

- DSSA users can use the generators to create specific components in the subdomain with far less effort.
- DSSA architects can use the generators to create reusable subsystems that can then form part of the component base available to DSSA users.

We have already identified the message handling subdomain as a candidate for AP5 technology; a tentative choice for the next area to tackle is fusion processing.

Figure 2 shows the activity flow that will be followed: identifying classes of components (subdomains) to be addressed, based on the architecture; defining domain specific languages and producing generators; developing annotations to permit optimization; and generating reusable application components.

C2 Message Handling

As indicated in Figure 3, the message handling subsystem is one of the key interfaces between a C2 system and the "outside world". It provides a means of communicating information between different C2 systems and to/from other C2 resources (such as vehicles and weapon installations). Messages may be text or bit streams; we will deal here with text messages. Some text messages are free-form, but most today follow standard prescribed formats; we will deal with formatted messages.

C2 messages are created by humans (on the transmitting

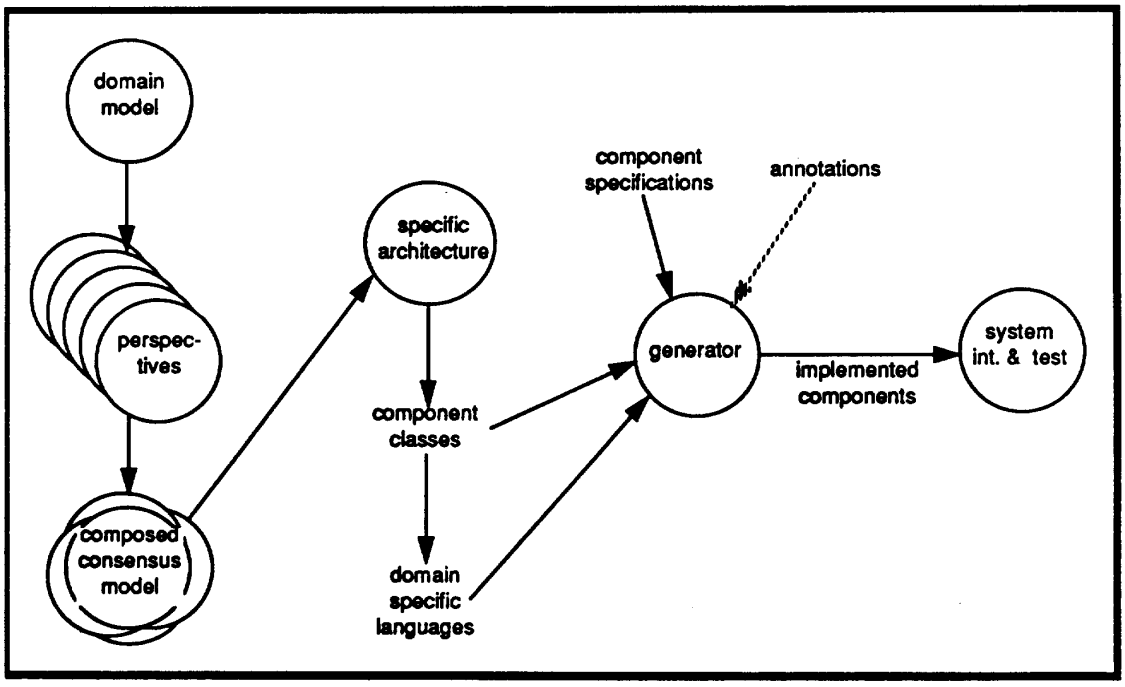

Figure 2. DSSA Application Generation Activity Flow

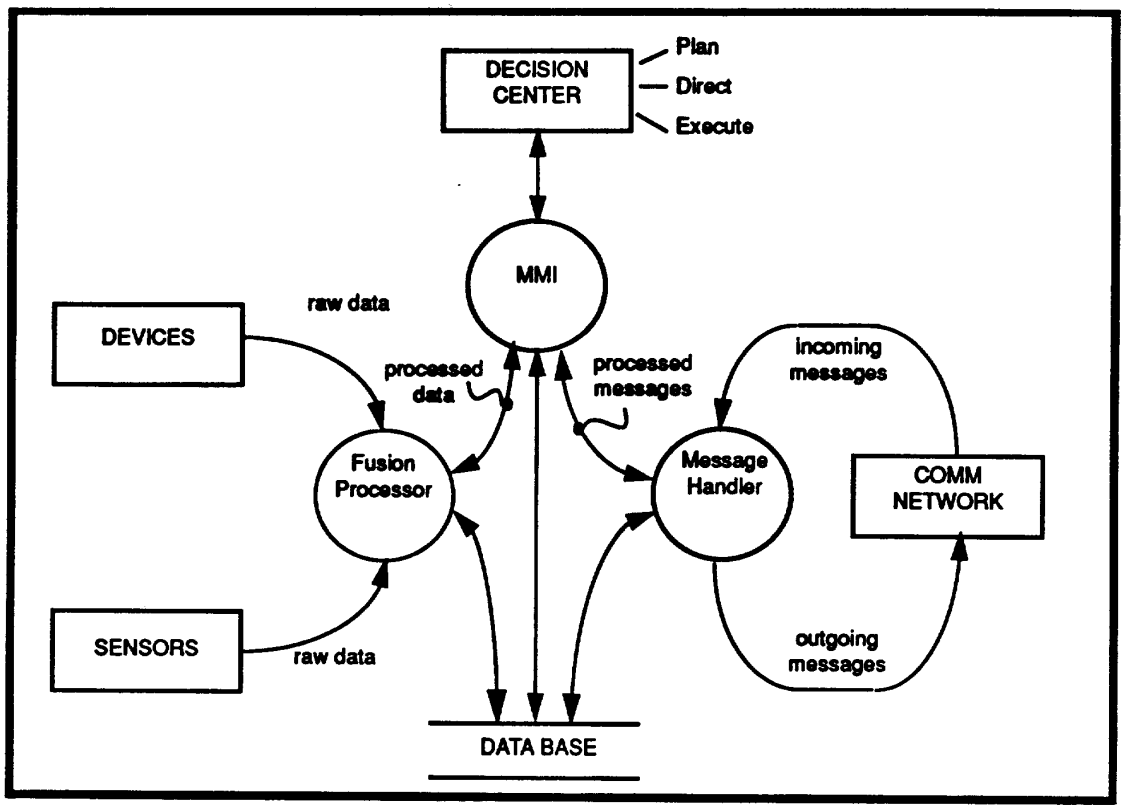

Figure 3. C2 System Operations

side of the interface) according to a written description of the formats. The receiving side parses the message (according to an encoded understanding of the standard format), validates it for correctness, and places the received information in the database for use by other parts of the system (for example, decision support).

There are several standard families of messages, for example NATO and JINTACCS messages. Each of these can include several hundred message types; for example, there are approximately 300 NATO message types. (Many types of messages are shared by several message families.) Message formats are described in massive documents using *ad hoc*, non-standard description methods. Typically the descriptions involve much prose description. For example, Figure 4 shows the description for a single line in one type of message. Furthermore, it is not a complete description; many field descriptions cross-reference to other descriptions.

A message consists of a number of such lines (called *datasets* -- may be more than one physical line) grouped together in an *envelope* (which contains from/to information, classification level, etc.). While each type of message can contain only certain kinds of datasets, many are optional and their order is generally not prescribed (though there are exceptions). Validity of datasets can depend on other datasets in the message. Each dataset contains a prescribed sequence of *fields*, separated by slashes, with a required order and a well-defined format. Field validity can depend on values in other fields of that dataset as well as in other datasets in the message. Figure 5 is an example message (excluding the envelope).

The code involved in writing the software to implement message handling is extensive and error prone. Working from the prose specification, programmers write code to extract each field from each dataset, validate it according to the specified rules, translate it to the appropriate internal representation, build database update transactions, and write to the database. Typically, a single message type can take from 5000 - 100,000 lines of HOL code. The Navy WWMCCS system uses approximately 4 million lines of code to implement 30 message types. Clearly this is a part of C2 system development that should be considered for automation.

Data Set ID: MSGID

Fld	Element Descriptive Name	Descript.	M	Edit Rule	Remarks
1	Message Code Name	25 AN	x	1. Must be a member of the approved set of message code words.	
2.	Originator	25 AN	x	1. Must be a plain language address or approved short title	a. Plain language address are validated against values found in the references
3.	Message Serial Number	3 N	a	1. Positive integer between the values 001 to 999.	a. May be required for specific messages.
				2. Out of sequence may indicate missing message. See rules for specific msg. code word.	b. Sequence is restarted on 1 Jan each year. May be rolled over when upper limit is reached. c. For Command authorities serial may be validated to maintain order when processing reports.
4.	As-of-Month	3 AN	a	1. Standard abbreviation for month message sent.	a. Required if serial number is used and as-of-DTG not present. b. Not allowed if as-of-DTG not present.
5.	As-of-Year	4 N	o	1. May not be a future year.	a. As-of-month must be present.

Figure 4. Example Message Line Description

```
NATOUNCLASSIFIED
SIC: NSR
EXER /OPEN GATE 91//
MSGID /NAVSITREP/CINCIBERLANT/135/ DEC/91//
PART /I/HOSTILE//
FORCE /OR523/3/37000N0-012000W3/145/17K/H//
SHIP /OR523A/KARA/-/CG/-/UR//
SHIP /OR523B/KRESTA//
SHIP /OR523C/KRESTA//
SUBTK /OR734/33000N6-010000W1/095/9K/M//
SUB /OR734/TANGO//
PART /II/UNKNOWN/NC//
PART /III/FRIENDLY//
FORCE /CTU 405.1.2/5/420015N2-1333440W8/175/20K//
FORCE /CTU 387.3.2/2/36010N0-004380W5/090/5K//
AMPN /MINE SWEEPING GROUP...//
AIRTK /934/33000N6-010000W1//
AMPN /ONE P-3 SEARCHIN BOX...//
```

Figure 5. Example Formatted Message

Automating C2 Message Handling Using AP5

To automate C2 message handling using AP5, we have developed a language specific to the message handling subdomain that provides constructs for specifying message formats, for indicating required validations, and for describing desired database updates.

Specifying Message Formats

Message formats are described in a simple set language that indicates which datasets are allowed and which are optional for a particular message type. For example,

```
type SPOT = (FORCE), (SHIPTK | AIRTK | AIRCRAFT),
            SHIP
```

would indicate that a SPOT message consists of an optional FORCE dataset, followed by an optional occurrence of one of the SHIPTK, AIRTK, or AIRCRAFT datasets, followed by a required SHIP dataset.

Message format descriptions can be accompanied by *validations* that indicate which combinations of datasets are valid. For example,

```
type SPOT = (FORCE), (SHIPTK | AIRTK | AIRCRAFT),
            SHIP
   validations
      disallow MSGID.message-serial-number;
      require SHIP.location
      no SHIPTK and no AIRTK requires FORCE;
```

indicates that the message-serial-number field of the MSGID dataset must not be present, the location field of the SHIP dataset must be present, and, if no SHIPTK dataset and no AIRTK dataset is present, the FORCE dataset must be present.

Specifying Datasets

Dataset formats are described in terms of the fields that make up the dataset and the format of each of those fields. Fields are ordered, so each dataset is characterized by a sequence of fields. Optional fields are indicated by parenthesizing them. Mutually exclusive fields are indicated by alternative bars. As for message formats, dataset descriptions can include validations. For example, a dataset description of a MSGID dataset might be:

```
dataset MSGID = message-code-name (originator)
                (message-serial-number) (as-of-month)
                (as-of-year) (as-of-DTG)
   validations
      as-of-DTG precludes as-of-month;
      as-of-DTG precludes as-of-year;
      as-of-year requires as-of-month;
      message-code-name /= SPOT requires originator;
      message-serial-number and no as-of-DTG
                requires as-of-month;
   field message-code-name = A*26;
   field originator = A*25;
   field message-serial-number = N 3;
   field as-of-month - month;
   field as-of-year = N 4;
   -- as-of-DTG in form: DDHHMMZSMMMYY
   field as-of-DGT - day, hour, minute, (Z), SUM1, month,
             year;
   field SUM1 = N 1;
   field day = N2;
   field hour = N 2;
   field minute = N 2;
   field month = A 3;
   field year = N 2;
```

Specifying Database Transactions

The C2 message description language also includes a means for describing the transactions to be carried out for each received message. An example of a segment of such a specification is:

```
{insert msg_Orig_Sr (ORIGINATOR = PROSIGN.FN,
   MSG_TYPE - MSGID.Code,
   MSG_DTG = sortable_data (ENVELOPE.DTG),
   CLASSIFY = classification_code(ENVELOPE.Sec));
   ...
```

The database update language also includes tests of field values, so that updates can be conditional on those values, and a capability to allow a sequence of updates to be named and reused in other update instructions. This simple language provides all the power needed to describe the database transactions resulting from received messages.

Implications

Clearly, automated generation of message handling software can save greatly on the labor involved in creating such software. A message handling subsystem that requires 4 million lines of HOL code should require less than 1% of that in the message description language.

Perhaps more significantly, there will be little reason to write most of the code more than once. The code required to parse and validate a message of a particular type is not specific to the system being implemented. Once the message specification is developed in the message description language, it can be reused. Minor changes in the specification of required database updates can be easily implemented for individual systems.

An even more far-reaching impact of this work is the development of a precise, unambiguous way of describing message formats. Rather than the *ad hoc* prose descriptions now used in describing message formats, the message description language can be used directly. This will eliminate errors in understanding and correctly implementing message descriptions.

This precise message description mechanism, along with the built-in incentive to reuse message description implementations, will contribute substantially to the development of more error-free message handling subsystems. A major aspect of this benefit is improved interoperability, as systems will no longer be dependent on the programmers' understanding of message formats. All implementations will share a common understanding and be able to interoperate with the full power and precision envisioned for formatted messages.

Acknowledgment

The work described in this paper has been supported by the Defense Advance Research Projects Agency through U.S. Army Communications-Electronics Command Contract No. DAAB07-92-C-Q502 and through NASA Ames Research Center Contract No. NCC 2-520.

References

[1] Balzer, Bob and Martin Feather, Neil Goldman, Dave Wile, "Proposal for DS Languages for C3 Messages," USC/ISI working paper, 1992.

[2] Braun, Christine L. and William L. Hatch, "Software Reuse Through CCIS Architecture Standardization," *Proceedings of the 11th AFCEA Europe Symposium and Exposition*, October 1990.

[3] Hatch, William, "Example Message Descriptions and Database Transactions," GTE working paper, 1992.

[4] Ruegsegger, Theodore, "Domain Specific Software Architectures -- Command and Control," briefing slides, CECOM Real-Time/ Reuse Technical Interchange Meeting, Ft. Monmouth, NJ, February 1992.

[5] Wile, David S., "Adding Relational Abstractions to Programming Languages," *Proceedings of workshop on Formal Methods in Software Engineering*, Napa Valley, CA, May 1990.